# SINC-LINK

# TIMEX-SINCLAIR USERS CLUB

# NEWSLETTER

## Toronto, Ontario



PRINTED IN CANADA

B6146

## MESSAGE FROM THE PRESIDENT

As you all know or may not know, the present executive was appointed to serve an interim of six months. That period is up now and some of us are relocating our directions. As such, I will be stepping down as President and a new one, George Chambers, will be coming in. George has done a fine job co-ordinating in the past year and we wish him all the best in the future.

I will probably be in one of the other executive positions and if not, then I hope that I can help the new Executive in any way possible.

Regards,
Greg Lloyd

## EDITOR'S NOTE

I'm sure I speak on behalf of the rest of the membership, Greg. You have done a fine job as president and the club has come a long way since our 'Get-Together' days. Even though the club is now more formalized, we have not lost sight of the 'free-for-all' or a one-on-one member contact during our meetings which I'm sure has helped a lot of members with some difficult areas of the computer.

Congradulations, George, as the new President of Timex-Sinclair Club.Good luck as our new President.

## EXECUTIVE OFFICERS

PRESIDENT: Greg Lloyd
SECRETARY: George Chambers
LIBRARIAN: Martin Mauk
TREASURER: John Roach
NEWS EDITOR: Stan Piotrowski
ACTIVITY DIRECTORS: Brian Hammond, Ian Roberts
MEETING CHAIRMAN: Harold Goodwin
LIASON OFFICER: (out-of-town members) Chris Hart

## CHECKING COMPUTER ACCURACY

(by Bill White)

An article in the April issue of SCIENTIFIC AMERICAN gives a method for determining the calculating precision level of a computer or calculator. The procedure is to enter 1.0000001 and square it 27 times. The correct result is, to ten digits,

534,339.4707

but most machines will be wrong.

All calculators can usually have the first three digits correct so that the per cent error is under 0.1. Unfortunately, most computers are a rather sad lot when tested on this method. These are some of the results:

| COMPUTER | RESULT | %ERROR |
|---|---|---|
| Apple II Basic | 22723.9709 | 96.6 |
| IBM PC Basic | 8850273. | 1212. |
| Ontel Basic | 8886690 | 1217. |
| ZX-81 Basic | 710493.46 | 5.3 |

The comparison with these others, makes our little "toy" look like a giant. So don't knock it!

The program I used to test the ZX-81, which listed the numbers as they were calculated is:

```
10  INPUT A
20  FOR B = 1 TO 27
30  LET C = A ** 2
40  PRINT C;" ";
50  LET A = C
60  NEXT B
```

## A MACHINE CODE PROGRAM WITH 1 POP

(by K. Van Vliet)

This is the Machine Code program I used to discuss turning a Call PQ Routine (a GOSUB with a RETURN) into a GOTO PQ with the RETURN address:
Q + 1 + 256 P in HL as follows:
First, set up the following Basic program, RUN it and then LOAD all the decimal values found at the end of this article when prompted for each INPUT N at its proper address:

NOTE: If you made an error then iput a number larger than 255 and start over.

```
 1  REM ...25 characters minimum...
10  FOR I = 16514 TO 16538
20  PRINT I;" ";
30  INPUT N
40  PRINT N
50  POKE I,N
60  NEXT I
70  CLS
80  PRINT AT 5,0;"inverse space"
90  RAND USR 16523
100 POKE 16528,118
110 POKE 16532,118
120 PRINT AT 10,0;"inverse space"
130 RAND USR 16523
140 STOP
```

| ADDRESS | OPERATION | HEX | DEC |
|---|---|---|---|
| 16514 | POP HL | E1 | 225 |
| 16515 | LET A=PEEK HL | 7E | 126 |
| 16516 | CP 128 WITH A | FE80 | 254,128 |
| 16518 | RETURN IF ZERO | C8 | 200 |
| 16519 | RST 10 (PRINT) | D7 | 215 |
| 16520 | INC HL | 23 | 35 |
| 16521 | GOTO (-8) | 18F8 | 24,248 |
| 16523 | CALL 16514 | CDPQ | 205,130,64 |
| 16526 | K | | 48 |
| 16527 | . | | 27 |
| 16528 | | | 0 |
| 16529 | V | | 59 |
| 16530 | A | | 38 |
| 16531 | N | | 51 |
| 16532 | | | 0 |
| 16533 | V | | 59 |
| 16534 | L | | 49 |
| 16535 | I | | 46 |
| 16536 | E | | 42 |
| 16537 | T | | 57 |
| 16538 | inverse space | | 128 |

## CORRECTION

In Newsletter Vol.2/No.2 or the March issue, I inadvertently made a mistake in basic lines 30 & 40 of page 12. The address you POKE your numbers into should be 16450 for line 30 and 16451 for line 40. The reason for using these 2 special locations was that it makes the machine code program re-locatable anywhere in memory. So if you want to load the REM generator at address 8192 (the 8K area if you have 64K or the Hunter Board), you can do this without changing any of the machine code. But you must change the Basic Line (after the above correction) of line 50. If you load the program at 8192 then you must use 'RAND USR 8192'. If you load it at address 10000, then 'RAND USR 10000'. In all cases the machine code will read the values stored at 16450 & 16451 which is 2 bytes of spare RAM memory not used by the computer.

## MACHINE CODE PROGRAMMING

Now that you have your 'feet a bit wet' in machine code, let's get on with just a few more commands. If we want to print a character to the screen, there are a few ways to do it. The simplest way is a one byte machine code command which prints any of the printable characters. If you look in the manual, you will see a few RST's which are actually machine code subroutines built within the Z-80 microchip. The one mostly used is RST 16d (or RST 10h) which is 'PRINT A CHARACTER ROUTINE'. (By the way, whenever you see the small letter 'd' or 'h', this simply means the number is 'decimal' or 'hexadecimal' or 'hex'.) There is only one stipulation with using RST 16 and that is the character you wish printed must be contained in the 'A' register and it must be printable. The machine code byte for RST 16 is 215d or D7h. Therefore, you must first load the 'A' register with a printable character in code form. Checking your manual again, you will find that the printable characters run from code '0' to code '63' and their

inverse is from code 128 to code 191.

As you recall in our earlier Newsletter, you must load the machine code somewhere (above RAMTOP, the 8K area, or from a REM statement.) and the most convenient place is from a REM statement which is always at 16514. So enter a REM statement with a few characters (at least 4); then using your loader program (or by direct command since it is a tiny program), POKE the following numbers in:

POKE 16514,62 POKE 16515,38 POKE 16516,215 POKE 16517,201.

This is what it looks like in mnemonics:

| LINE | CODE | MNEMONIC | COMMENTS |
|---|---|---|---|
| 1 | 62,38 | LD A,38 | load 'A' with the code for the letter 'A' |
| 2 | 215 | RST 16 | Call the subroutine to print a character |
| 3 | 201 | RET | return to Basic |

Now RAND USR 16514 and the letter 'A' is printed on the screen. This may not seem like a big deal now but it's the first step in machine code programming. Let's expand the program a little to show you the real value and speed of machine code.

The above program is similar to the following Basic:

```
10  LET A$="A"
20  PRINT A$
30  STOP
```

Expanding the machine code routine.....

| LINE | CODE | MNEMONIC | COMMENTS |
|---|---|---|---|
| 1 | 62,38 | LD A,38 | Load the 'A' register with the letter 'A' |
| 2 | 215 | RST 16 | Call the subroutine to print a character |
| 3 | 24,251 | JR 251 | Jump back 5 bytes (to start) |

Now, RAND USR 16514, and immediately, the whole screen fills with the letter 'A'. Try this with a few other character codes or their inverse. The above machine code is somewhat similar to adding line 30 to the above Basic line which is:

30 GOTO 10

There is still a faster printing routine but we'll get into that some other time. In the meantime, I will explain the new command: 'JR'. There really is no equivalent in Basic. In the above Basic line, we have what would be considered in machine code as 'absolut addressing'. In other words, the program must GOTO a specific line and no others. The machine code equivalent would be a 'JP' (or Jump) command which would be Jump to a specific address. But the machine code command 'JR' means 'Jump Relative' to the address it is on to another address. In this case it was 5 bytes. Notice, no specific address is specified; only the number of bytes to jump. What this means is that by jumping relative, you save 1 machine code byte (since JP takes a total of 3 bytes), and most important is that the whole machine code program is re-locatable anywhere in memory. In the JR routine,

no specific address is specified which means that it only jumps forward or backward, the required number of bytes. Threfore, you could load the above routine at 8192 then RAND USR 8192 and get the same results.

Again, you must learn a new rule or exception to the above 'JR'. You are restricted as to how far you can jump relative; you can only jump ahead 128 bytes or back 127 bytes. If the jump is greater, then you must use a 'JP' (jump to absolute address) instead. Therefore, when the computer comes across the machine code command '24', the byte following tells the computer how far to jump and whether it is a forward or a backward jump. If the computer reads a number from 1 to 128, then the routine will jump ahead that many bytes; but if it sees a number from 129 to 255, then that means to jump backwards. But how do we determine the number of bytes backwards? Forward is simple enough since we just read it directly; for the backwards jump, subtract that number from 256 and the result is the number of bytes to jump back. Therefore, in the previous machine code routine, JR 251 means 256-251=5 which is jump back 5 machine code bytes.

There is something else to remember when figuring the number of bytes (this is usually the most troublesome area of machine code). Whenever determing the number of bytes forward or backwards, ALWAYS REMEMBER TO COUNT THE BYTE ITSELF. For example, if the computer reads JR 3, and the bytes in the program were: 10,16,24,3,1,18,22,64; it would jump to the address hold ing the byte value 64. Let's go through this step by step because anytime you have trouble in machine code, always check the JR routines.

The computer reads the first 2 bytes then sees the byte 24. It next reads how many bytes to jump relative and the following byte is 3. Therefore, the computer has processed bytes 10,16,24 and no it is processing byte value #3. Jumping ahead 3 bytes, the first byte to jump is 1, the next byte is 18 and the 3rd byte to jump is 22. The computer then begins its machine code command execution at byte value 64 which is the 4th byte (3 bytes have been jumped). If you check back on our previous machine code tiny program, you should see how we arrived at jumping back 5 bytes. Don't forget, the computer first reads the command for JR then the number of bytes which are two bytes processed before the actual command is executed.

Again, if we are outside of this range, then then you must use a JP instruction (or if you're very careful, another JR instruction). The JR instruction uses only 2 bytes whereas the JP instruction uses 3 bytes. For example, in the previous machine code program, you could have used JP 16514 (jump to the address of 16514) which would be in machine code: 195,130,64. But you can readily see that if you decided to move this to the 8K area, then you would have to change the JP address which instead of 16514 would be 8192 which means more math calculations.

You now have 2 more new commands learned: JP and JR. But again there are duplications (of a sort). If you check your manual, the code 40,n which is JR z,n and code 202; JP z,nn. These mean if the arithmetic performed equals zero then execute a JR or a JP. There are other commands similar such as JR C or JP C (carry set-to be discussed), etc. With the 2 new commands, you have learned several new commands.

**Arithmetic Performed = 0**

The arithmetic used is a simple subtraction performed on the 'A' register and can be performed on the 'A' register only. If you check your manual, notice for the codes 184 through to 191...CP B, CP C, etc. (& code 254...CP n). But compare 'B' to what? You can only compare any of the registers or numbers to whatever value is contained in the 'A' register. So instead of writing: "CP B TO A", since we can only compare to the 'A' register, we write: CP B and 'A' is understood. For example, if the 'B' register contained the value 10 and the 'A' register contained the value 10, when the computer comes across the command 'CP B', it will subtract the 'B' register from the 'A' register. NOTE: it doesn't actually subtract the 2 registers. They will still contain the same values. It subtracts them 'mentally' if you like to determine the results.

After performing the subtraction, if the result is zero, then (in computer terminology) the zero flag is raised. If the computer came across the next instruction which might be JR Z, 10...it would jump ahead 10 bytes since the zero flag is set or raised.

We can compare any of the registers with 'A' or a number from '0 to 255' with the 'A' register. This is a very important and widely used command on the computer. For example, if in a space invaders game you were allowed to be hit 3 times before GAME OVER, the computer could decrease the value by one each time you were hit and after the third time, jump to the routine (JP Z) to tell you the game is over.

The above raises a new and another important question; that is, the meaning of the FLAGS. The ones mostly used are:

**C - CARRY  NC - NO CARRY  Z - ZERO  NZ - NOT ZERO**

There are more but you will very seldom use them and by the time you want to use them, there are books to explain them and you are well on your way in machine code almost to the expert level.

Back to the flag issue. If you recall, a byte is made up of 8 bits: e.g. 10101010. If the bit is a '1' then that bit is said to be set. If the bit is a '0', then the bit is said to be reset. Somewhere out in the far reaches of the computer, 1 byte (i.e. 8 bits), are set aside for the FLAGS. Each bit in this byte is reserved for 1 function: 1 bit is for the zero flag, another for the carry, etc. Since we cannot directly access the bits, it's not worth mentioning which bit does what function. Since we only want to know whether a certain bit is set (flag raised), or reset (not zero for example), then we put in that machine code command and the computer checks the proper bit for us. Therefore, if the computer comes across the command: 'JR Z', it checks the Zero flag; if it is set (raised), then it will execute the JR command; if it is reset (flag not raised or bit is zero), then it does nothing with the command after checking the zero flag.

Even though some books show which bit represents which flag, that is of no concern to us since the computer does all the checking. But remember, 2 flags may be affected by the 'CP' command. For example, if the 'B' register contained 20 and the 'A' register contained 10 and by doing a comparison we subtract 20 from 10. Obviously this does not equal zero so

the zero flag is lowered (bit=0 or zero flag reset). But since the computer deals with whole positive numbers only, we have what is called an overflow and the 'CARRY' bit is set (or raised). We must keep this in mind if we will be concerning ourselves later in a machine code program of checking both the 'carry' and the 'zero' flags.

The carry flag is simple to understand. We knoe that if we subtracted 10 from 10, we get zero or if we subtract 5 from 10, we get 5. Notice that they are all in the range from '0-255' and are positive numbers. Therefore, if we subtract 20 from 10, in reality we get minus 10 (-10) or negative 10. We kno that the computer deals in positive whole numbers only so that the only way of passing this information on to us is by the computer setting the 'CARRY' bit (or raising the 'carry' flag). The same thing would happen if the 'A' register contained the value 255 and it came across the instruction: 'ADD A,10'. Again, we would have an 'overflow' (numbers gone past the '0-255' range). There actually is no overflow as such. It simply reverts back to zero and starts counting again. But we need some way of knowing that the number is greater than the value capable of being held by the register. This is done by setting the 'carry flag'. Therefore 'A' would now contain the value '9'. Remember, '0' is a number too.

This can be useful in general addition. Don't forget, all calculators are like computers. The arithmetic to be performed are simply a set of machine code instructions. For example, if we were to add 2 numbers that go beyond the 255 range, then the first register will contain the new number and we can 'INC' the second register by '1' if 'carry is set'.

If you have understood this article, you will have learned 3 new commands (which again are duplicated in a slightly different form). These commands and a complete understanding of the flags are very important in machine code programming since they are used quite extensively. On page 59 of 'Toni Bakers' book (MASTERING MACHINE CODE ON YOUR ZX-81), she shows a routine on how to print a message to the screen. Go through the routine and disregarding her unsual commands such as 'CALL SPRINT', check the machine code instructions on what she is doing. This particluar routine has been used by me on several different machine code programs...even for a game since you can print the whole graphic game in machine code using her routine before the actual playing of the game. (This of course will be explained further in a future article.) To understand any of her routines or your own, do the same thing I do; read the machine code instruction and using a scratch pad, execute each instruction as the computer would and write down the results. For example, if you LD A, 10, then write down A=10. The next instruction might be INC A. Cross out 10 and write 11. This is the only we I have found to check why a particluar machine code routine wouldn't work and finding my errors since in machine code, we have no SYNTAX checking.

That's it for now so until next time, have fun programming in machine code. Remember, it may take 10 times longer thanBasic to program in machine code but the program will work hundreds of times faster. Very impressive.

```
   1 REM NEWSLETTER NAME FILE
   2 REM BY STAN PIOTROWSKI
   3 REM JANUARY  1984
   4 REM
   5 REM
  10 GOTO 9000
  20 PRINT R$(I,1 TO 26)
  22 PRINT R$(I,27 TO 49)
  24 PRINT R$(I,50 TO 57)
  26 PRINT R$(I,58 TO 69)
  28 PRINT R$(I,70 TO 76)
  30 PRINT R$(I,77 TO 88)
  32 RETURN
 100 FAST
 110 CLS
 120 PRINT AT 6,6;"NAME/ADDRESS
FILE"
 130 PRINT TAB 5;"
":
 140 PRINT ,,TAB 6;"1. ADD"
 150 PRINT TAB 8;"2. DELETE"
 160 PRINT TAB 8;"3. SORT"
 170 PRINT TAB 8;"4. LIST"
 180 PRINT TAB 8;"5. SEARCH"
 190 PRINT TAB 8;"6. SAVE"
 200 PRINT ,,TAB 7;"ENTER A NUMB
ER"
 210 SLOW
 220 IF INKEY$<"1" OR INKEY$>"6"
 THEN GOTO 220
 230 GOTO 1000+VAL INKEY$
1000 FAST
1010 CLS
1020 LET N=N+1
1030 IF N>51 THEN GOTO 9600
1040 PRINT ,,,,,"ADD NEW RECORD T
O FILE"
1050 PRINT L$
1060 PRINT ,,,,,"NAME:  (SURNAME F
IRST)"
1065 SLOW
1070 INPUT N$
1080 PRINT AT 6,6;N$
1090 PRINT ,,"ADDRESS:  ";
1100 INPUT A$
1110 PRINT A$
1120 PRINT ,,"CITY/TOWN:  ";
1130 INPUT T$
1140 PRINT T$
1150 PRINT ,,"POSTAL CODE:  ";
1160 INPUT C$
1170 PRINT C$
1180 PRINT ,,"PHONE NUMBER:  ";
1190 INPUT P$
1200 PRINT P$
1210 PRINT ,,,,,TAB 6;"ALL CORREC
T? (Y/N)"
1220 IF INKEY$="N" THEN GOTO 130
0
1230 IF INKEY$="Y" THEN GOTO 150
0
1240 GOTO 1220
1300 FAST
1310 CLS
1320 PRINT AT 6,6;"RECORD ";N;"
CANCELLED"
1330 PRINT ,,,,,TAB 9;"PRESS A KE
Y"
1340 PAUSE 4E4
1350 LET N=N-1
1360 GOTO 1000
1500 FAST
1510 CLS
1520 LET R$(N,1 TO 26)=N$
1530 LET R$(N,27 TO 49)=A$
1540 LET R$(N,50 TO 69)=T$
1550 LET R$(N,70 TO 76)=C$
1570 LET R$(N,77 TO 88)=P$
1580 PRINT AT 8,9;"ADD ANOTHER R
ECORD? (Y/N)"
1590 PAUSE 4E4
1600 IF INKEY$="Y" THEN GOTO 100
0
1610 GOTO 100
2000 FAST
2010 CLS
2020 PRINT AT 6,0;"ENTER RECORD
NUMBER TO DELETE:"
2030 SLOW
2040 INPUT X$
2050 FAST
2060 IF X$<"1" OR X$>STR$ N THEN
 GOTO 2200
2065 CLS
2066 LET I=VAL X$
2070 GOSUB 20
2075 PRINT ,,"IS THIS THE RECORD
 TO DELETE? YN"
2076 PAUSE 4E4
2077 IF INKEY$="Y" THEN GOTO 208
0
2078 GOTO 100
2080 LET Z=VAL X$
2090 FOR I=Z TO N-1
2100 LET R$(I)=R$(I+1)
2110 NEXT I
2120 LET R$(N)=""
2130 LET N=N-1
2140 PRINT AT 10,0;"RECORD ";X$;
" HAS BEEN DELETED"
2150 GOTO 2230
2200 FAST
2210 CLS
2220 PRINT AT 6,2;"RECORD ";X$;"
 IS NOT ON FILE"
2230 PRINT AT 20,2;"DELETE ANOTH
ER RECORD? (Y/N)"
2235 PAUSE 4E4
2240 IF INKEY$="Y" THEN GOTO 200
0
2250 GOTO 100
3000 FAST
3010 CLS
3020 FOR P=N-1 TO 1 STEP -1
3030 FOR Q=1 TO P
3040 IF R$(Q,1 TO 26)<=R$(Q+1,1
TO 26) THEN GOTO 3100
3050 LET W$=R$(Q)
3060 LET R$(Q)=R$(Q+1)
3070 LET R$(Q+1)=W$
3100 NEXT Q
3110 NEXT P
3120 PRINT AT 8,6;"ALL RECORDS S
ORTED"
3130 PRINT ,,TAB 6;"PRESS A KEY
FOR MENU"
3140 PAUSE 4E4
3150 GOTO 100
4000 FAST
4010 CLS
4020 LET I=1
4030 PRINT "NO. ";TAB 9;"NAME"
4040 PRINT L$
4050 PRINT I;TAB 4;R$(I,1 TO 25)
4060 LET I=I+1
4070 IF I>N THEN GOTO 4400
4080 IF PEEK 15442>4 THEN GOTO 4
050
4085 PRINT AT 21,0;"C=CONTINUE,
P=PRINTOUT, M=MENU"
4090 PAUSE 4E4
4100 IF INKEY$="P" THEN GOTO 420
0
4110 IF INKEY$="C" THEN GOTO 430
0
4120 IF INKEY$="M" THEN GOTO 100
4130 GOTO 4090
4200 COPY
4210 GOTO 4090
4300 CLS
4310 GOTO 4030
```

```
4400 PRINT AT 20,4;"NO MORE RECO
RDS ON FILE"
4410 GOTO 4085
5000 FAST
5005 LET G$="N"
5010 CLS
5020 PRINT AT 8,0;"ENTER NAME TO
 SEARCH"
5022 SLOW
5025 INPUT K$
5030 LET S$=K$+"
          "
5032 LET S$=S$(1 TO 26)
5035 FAST
5040 LET I=1
5050 IF S$=R$(I,1 TO 26) THEN GO
TO 5100
5060 LET I=I+1
5070 IF I>N THEN GOTO 5090
5080 GOTO 5050
5090 CLS
5094 PRINT AT 20,4;"PRESS A KEY
FOR MENU"
5095 PRINT AT 12,(31-LEN K$)/2;K
$
5096 IF G$="Y" THEN PRINT AT 8,4
;"NO MORE MATCHES FOR"
5097 IF G$="N" THEN PRINT AT 8,8
;"NO MATCHES FOR"
5098 PAUSE 4E4
5099 GOTO 100
5100 CLS
5105 PRINT AT 3,5;"RECORD NUMBER
";I
5106 PRINT ;;;;
```

```
5107 LET G$="Y"
5110 GOSUB 20
5120 PRINT AT 20,0;"P=PRINTOUT,
 C=CONTINUE, M=MENU"
5130 PAUSE 4E4
5140 IF INKEY$="P" THEN GOTO 520
0
5150 IF INKEY$="M" THEN GOTO 10
5160 IF INKEY$="C" THEN GOTO 530
0
5170 GOTO 5130
5200 COPY
5210 GOTO 5120
5300 CLS
5320 LET I=I+1
5330 GOTO 5050
6000 FAST
6010 CLS
6020 PRINT AT 8,0;"ENSURE TAPE R
ECORDER IS ON"
6030 PRINT ,,"THE RECORD MODE TH
EN PRESS A KEY"
6040 PAUSE 4E4
6050 SAVE "NAME FILE"
6060 GOTO 100
9000 DIM R$(50,88)
9010 DIM N$(26)
9020 DIM A$(23)
9030 DIM T$(20)
9040 DIM C$(7)
9050 DIM P$(12)
9060 DIM U$(88)
9070 DIM S$(26)
9100 LET N=0
9110 LET L$="----------------------
----------------"
9200 GOTO 100
```

The above program is the one we have been discussing in the earlier Newsletters. Since some of the program lines were not clear and we have some new members, I thought that you would like to see the program in its entirety. For those of you who have been learning along since the beginning, the last set of lines to add are the ones starting at 3000. This deals with SORTING the files alphabetically. The routine used is a simple "Bubble Sort". I'll explain how it works and you can go through the algorithms to see the logic.

Basically, the sort compares the 2 consecutive records character by character and the first time it encounters a difference, it checks to see which has a higher value. If the first record has a higher value than the second one, an exchange is made. (See lines 3040 to 3070). Since the computer cannot make a straight exchange, it assigns the first record number to a different variable; it then assigns the second varibale to the first record; lastly, it assigns the second record to equal the new or floating variable. It continues on up until it finds the end then starts over again until each and every record has been compared with each other. (The record 'BUBBLES' up to the top.).

If you check the 'Character Set' in the back of the ZX-81 manual, you will see that each character is assigned a decimal value. For example, the number 2 is assigned the value 30 while the letter 'A' is assigned the value 38. When sorting, the computer converts the record characters to numbers and compares the numbers; if the number it is comparing is a higher value, then the records are switched around. That's the reason why a record with a number precedes a record with a letter as their first characters.

That's it for now on Basic programming. The following Newsletters will deal with expanding this program to include partial record searches and how to change the program to be something else such as cataloging magazine articles or record albums. You now have the "Basic Skeleton" so you only have to change the titles and/or DIMensions to suit your needs.

## LOW MEMORY:- SAVE/LOAD PROGRAM

IF YOU HAVE 64K AND USE THE 8192/19383 (2000/3FFF)AREA, THIS WILL HELP YOU SAVE AND LOAD IT.

IT IS A REVISION OF TONI BAKERS HEXLD TO KEEP RAM TOP AT 64K. YOU CAN ADD YOUR OWN HEX LOADER OR POKE IN THE VALUES. THIS EXAMPLE IS TO SAVE MY PROGRAM FROM 335F TO 3FFF. I USE THAT AREA FOR:-

| | |
|---|---|
| H. PRINT | TOOL KIT |
| H. LIST 1 | H. LIST 2 |
| S. PRINT | D. PRINT |
| 44.7K VAR- | MEM:- |
| STORE | USED |
| SEARCH | SPARE |
| PRINT | RAM TOP |

BOB CROKER 3/28/84

```
 10 REM ?NSE RNDE RND GOSUB ?R
ND GOSUB ?6 RND??TAN E(RND) ;
FOR E RND GOSUB ?RND GOSUB TAN
 E(RND) ; GOSUB ?RND GOSUB ?R
ND GOSUB TAN
 400 DIM O$(USR 16520)
 410 RAND USR 16536
 420 SAVE "SAVE L/O MEM."
 500 RAND USR 16554
 510 CLEAR
 520 STOP
 525 REM - IF LO MEM REV. THEN
CHANGE THE POKE
 526 REM - NOTE:- (LIMIT) IS ONE
 MORE THAN END OF M/C PGM.
 530 REM POKE 16514, LO (BEGIN)
 531 REM POKE 16515, HI (BEGIN)
 532 REM POKE 16518, LO (LIMIT)
 533 REM POKE 16519, HI (LIMIT)
 535 REM - RUN TO SAVE L/O MEM.
 536 REM - USE GOTO 9900 TO SAVE
 THIS PROGRAM
9900 SAVE "SAVE PGM"
```

| | | | |
|---|---|---|---|
| 16514 | 95 | 16543 | 235 |
| 16515 | 51 | 16544 | 42 |
| 16516 | 161 | 16545 | 130 |
| 16517 | 12 | 16546 | 64 |
| 16518 | 0 | 16547 | 237 |
| 16519 | 64 | 16548 | 75 |
| 16520 | 42 | 16549 | 132 |
| 16521 | 134 | 16550 | 64 |
| 16522 | 64 | 16551 | 237 |
| 16523 | 237 | 16552 | 176 |
| 16524 | 91 | 16553 | 201 |
| 16525 | 130 | 16554 | 42 |
| 16526 | 64 | 16555 | 16 |
| 16527 | 167 | 16556 | 64 |
| 16528 | 237 | 16557 | 17 |
| 16529 | 82 | 16558 | 6 |
| 16530 | 34 | 16559 | 0 |
| 16531 | 132 | 16560 | 25 |
| 16532 | 64 | 16561 | 237 |
| 16533 | 68 | 16562 | 91 |
| 16534 | 77 | 16563 | 130 |
| 16535 | 201 | 16564 | 64 |
| 16536 | 42 | 16565 | 237 |
| 16537 | 16 | 16566 | 75 |
| 16538 | 64 | 16567 | 132 |
| 16539 | 17 | 16568 | 64 |
| 16540 | 6 | 16569 | 237 |
| 16541 | 0 | 16570 | 176 |
| 16542 | 25 | 16571 | 201 |

```
4082 5F          LD E,A
4083 33          INC SP
4084 A1          AND C
4085 0C          INC C
4086 00          NOP
4087 40          LD B,B
4088 2A8540      LD HL,(4086)
408B ED5B8240    LD DE,(4082)
408F A7          AND A
4090 ED52        SBC HL,DE
4092 228440      LD (4084),HL
4095 44          LD B,H
4096 4D          LD C,L
4097 C9          RET
4098 2A1040      LD HL,(4010)
409B 110500      LD DE,0006
409E 19          ADD HL,DE
409F EB          EX DE,HL
40A0 2A8240      LD HL,(4082)
40A3 ED4B8440    LD BC,(4084)
40A7 EDB0        LDIR

40A9 C9          RET
40AA 2A1040      LD HL,(4010)
40AD 110500      LD DE,0006
40B0 19          ADD HL,DE
40B1 ED5B8240    LD DE,(4082)
40B5 ED4B8440    LD BC,(4084)
40B9 EDB0        LDIR
40BB C9          RET
```